

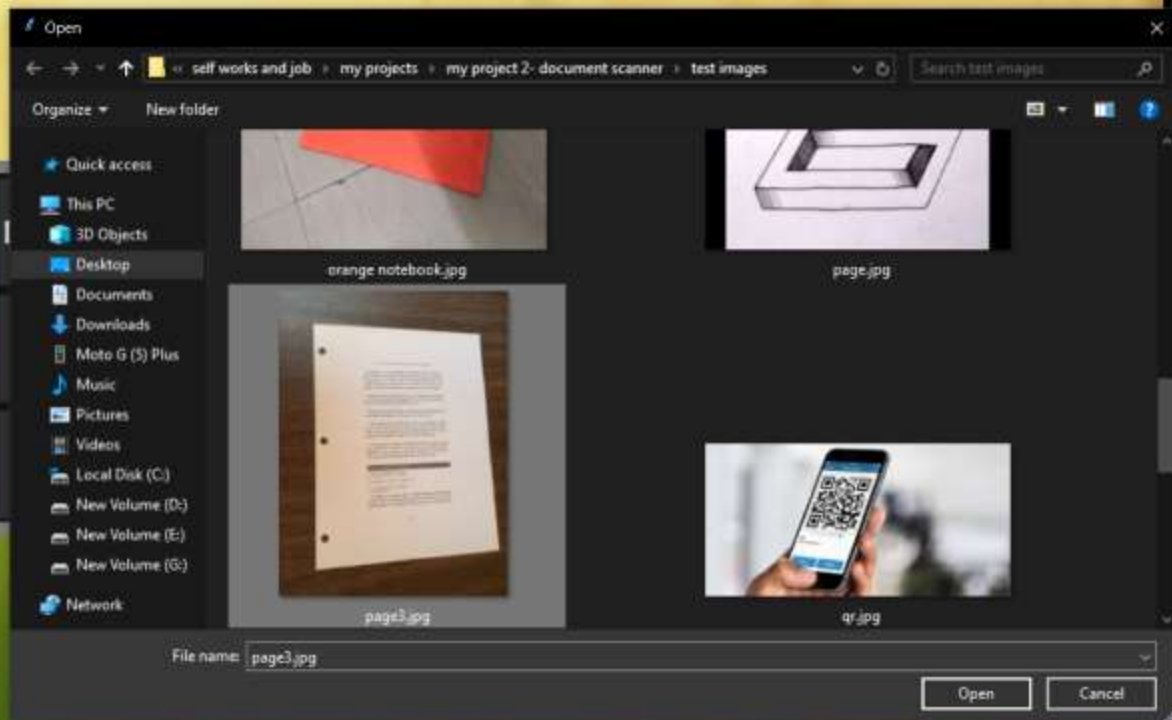
# Document Scanner Application

The background image shows a bright sun setting behind a field of tall, green grass. The sun is a bright white-yellow circle with rays emanating from it, partially obscured by a grass stalk in the foreground. The sky is a pale yellow, and the grass is a vibrant green. A dark grey menu box with a thin grey border is centered on the right side of the image.

Document Perspective Corrector

Image QR code detector

Exit



## 3.7 ACCESSING AND MANIPULATING PIXELS

On **Line 14** we manipulate the top-left pixel in the image, which is located at coordinate (0,0) and set it to have a value of (0, 0, 255). If we were reading this pixel value in RGB format, we would have a value of 0 for red, 0 for green, and 255 for blue, thus making it a pure blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, **not** RGB format.

We actually read this pixel as 255 for red, 0 for green, and 0 for blue, making it a red color, **not** a blue color.

After setting the top-left pixel to have a red color on **Line 14**, we then grab the pixel value and print it back to console on **Lines 15 and 16**, just to demonstrate that we have indeed successfully changed the color of the pixel.

Accessing and setting a single pixel value is simple enough, but what if we wanted to use NumPy's array slicing capabilities to access larger rectangular portions of the image? The code below demonstrates how we can do this:

## Listing 3-3: getting and setting.py

```
1 # Import cv2 module, as cv2
2 import cv2 as cv
3 # Read the image from disk
4 img = cv.imread('image.jpg', cv.IMREAD_COLOR)
5
6 # Grab the top-left 100 x 100 pixel region
7 roi = img[0:100, 0:100]
8
9 # Set the color of the ROI to blue
10 cv.cvtColor(roi, cv.COLOR_BGR2RGB)
11 cv.imshow('ROI', roi)
```

On **Line 17** we grab a 100 x 100 pixel region of the image. In fact, this is the top-left corner of the image! In order to grab chunks of an image, NumPy expects us provide four

Main Menu

Next

Rotate

Crop

## 4.3 ACCESSING AND MANIPULATING PIXELS

On **Line 14** we manipulate the top-left pixel in the image, which is located at coordinate (0, 0) and set it to have a value of (0, 0, 255). If we were reading this pixel's value in RGB format, we would have a value of 0 for red, 0 for green, and 255 for blue, thus making it a pure blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, **not** RGB format.

We actually read this pixel as 255 for red, 0 for green, and 0 for blue, making it a red color, **not** a blue color.

After setting the top-left pixel to have a red color on **Line 14**, we then grab the pixel value and print it back to console on **Lines 15 and 16**, just to demonstrate that we have indeed successfully changed the color of the pixel.

Accessing and setting a single pixel value is simple enough, but what if we wanted to use NumPy's array slicing capabilities to access larger rectangular portions of the image? The code below demonstrates how we can do this:

**Listing 4.3** *grabbing and setting pixels*

```
14 image = image[0:100, 0:100]
15 cv2.imshow('center', image)
16
17 image[0:100, 0:100] = (0, 255, 0)
18 cv2.imshow('updated', image)
19 cv2.waitKey(0)
```

On **Line 17** we grab a  $100 \times 100$  pixel region of the image. In fact, this is the top-left corner of the image! In order to grab chunks of an image, NumPy expects we provide four



## 4.3 ACCESSING AND MANIPULATING PIXELS

On **Line 14**, we manipulate the top-left pixel in the image, which is located at coordinate (0,0) and set it to have a value of (0, 0, 255). If we were reading this pixel value in RGB format, we would have a value of 0 for red, 0 for green, and 255 for blue, thus making it a pure blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, **not** RGB format.

We actually read this pixel as 255 for red, 0 for green, and 0 for blue, making it a red color, **not** a blue color.

After setting the top-left pixel to have a red color on **Line 14**, we then grab the pixel value and print it back to console on **Lines 15 and 16** just to demonstrate that we have indeed successfully changed the color of the pixel.

Accessing and setting a single pixel value is simple enough, but what if we wanted to use NumPy's array slicing capabilities to access larger rectangular portions of the image? The code below demonstrates how we can do this:

## Listing 4.3: getting and setting

```
17 corner = image[0:100, 0:100]
18 cv2.imshow('corner', corner)
19
20 image[0:100, 0:100] = (0, 255, 0)
21
22 cv2.imshow('blacked', image)
23 cv2.waitKey(0)
```

On **line 17** we grab a  $100 \times 100$  pixel region of the image. In fact, this is the top-left corner of the image! In order to

grab chunks of an image, NumPy expects we provide four

Reset



## 4.3. REDUCING THE NUMBER OF CALLS

On Line 14 we multiplied the top-left pixel in the image, which is located at coordinate (0,0), and set it to have a value of (0, 0, 0). If we were making this pixel value in RGB format, we would have a value of 0 for red, 0 for green, and 0 for blue. Here we'd set it to just blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, and RGB format.

We actually need this pixel as 000 for red, 0 for green, and 0 for blue, making it a red color, not a blue color.

After setting the top-left pixel to have a red color on Line 14, we then grab the pixel value and print it back to our side on Lines 15 and 16, just to be certain that we have indeed successfully changed the color of the pixel.

Accounting and setting a single pixel value is simple enough, but what if we wanted to set Numpy's array slicing capabilities to access larger, nonadjacent portions of the image? The code below demonstrates how we can do this:

## Listing 4.3.1: Setting the Color of

```
img[0,0,0] = 255  # Set the top-left pixel to red
img[0,0,0] = 0    # Set the top-left pixel to blue
img[0,0,0] = 0    # Set the top-left pixel to green
img[0,0,0] = 0    # Set the top-left pixel to red
```

On Line 17 we grab a 100 x 100 pixel region of the image. In fact, this is the top-left corner of the image. We provide four arguments to the image slicing operation.

In fact, this is the top-left corner of the image. We provide four arguments to the image slicing operation.

Main Menu

Next

Rotate

Done

Reset

## 4.3 ACCESSING AND MANIPULATING PIXELS

On **Line 14** we manipulate the top-left pixel in the image, which is located at coordinate (0, 0) and set it to have a value of (0, 0, 255). If we were reading this pixel's value in RGB format, we would have a value of 0 for red, 0 for green, and 255 for blue, thus making it a pure blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, **not** RGB format.

We actually read this pixel as 255 for red, 0 for green, and 0 for blue, making it a red color, **not** a blue color.

After setting the top-left pixel to have a red color on **Line 14**, we then grab the pixel value and print it back to console on **Lines 15 and 16**, just to demonstrate that we have indeed successfully changed the color of the pixel.

Accessing and setting a single pixel value is simple enough, but what if we wanted to use NumPy's array slicing capabilities to access larger rectangular portions of the image? The code below demonstrates how we can do this:

**Listing 4.3** *grabbing and setting pixels*

```
14 image = image[0:100, 0:100]
15 cv2.imshow('center', image)
16
17 image[0:100, 0:100] = (0, 255, 0)
18 cv2.imshow('updated', image)
19 cv2.waitKey(0)
```

On **Line 17** we grab a  $100 \times 100$  pixel region of the image. In fact, this is the top-left corner of the image! In order to grab chunks of an image, NumPy expects we provide four

## 4.3 ACCESSING AND MANIPULATING PIXELS

On **Line 14** we manipulate the top-left pixel in the image, which is located at coordinate (0,0) and set it to have a value of (0, 0, 255). If we were reading this pixel value in RGB format, we would have a value of 0 for red, 0 for green, and 255 for blue, thus making it a pure blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, **not** RGB format.

We actually read this pixel as 255 for red, 0 for green, and 0 for blue, making it a red color, **not** a blue color.

After setting the top-left pixel to have a red color on **Line 14**, we then grab the pixel value and print it back to console on **Lines 15** and **16**, just to demonstrate that we have indeed successfully changed the color of the pixel.

Accessing and setting a single pixel value is simple enough, but what if we wanted to use NumPy's array slicing capabilities to access larger rectangular portions of the image? The code below demonstrates how we can do this:

```
def grab_and_set_image():
    # Create a 100x100, 3 channel
    # BGR image
    img = np.zeros((100, 100, 3), dtype=np.uint8)

    # Grab a 10x10 region of the image
    img[0:10, 0:10] = 255, 0, 0

    # Set a single pixel to red
    img[0,0] = 255, 0, 0
```

On **line 17** we grab a  $100 \times 100$  pixel region of the image. In fact, this is the top-left corner of the image! In order to grab chunks of an image, NumPy expects we provide four

Main Menu

Next

Undo

Redo

Edge Detection

Smoothing

Spatial

Sharpening

Noise

## 4.3 ACCESSING AND MANIPULATING PIXELS

On **Line 14** we manipulate the top-left pixel in the image, which is located at coordinate (0,0) and set it to have a value of (0, 0, 255). If we were reading this pixel value in RGB format, we would have a value of 0 for red, 0 for green, and 255 for blue, thus making it a pure blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, not RGB format.

We actually read this pixel as 255 for red, 0 for green, and 0 for blue, making it a red color, not a blue color.

After setting the top-left pixel to have a red color on **Line 14**, we then grab the pixel value and print it back to console on **Lines 15 and 16**, just to demonstrate that we have indeed successfully changed the color of the pixel.

Accessing and setting a single pixel value is simple enough, but what if we wanted to use NumPy's array slicing capabilities to access larger rectangular portions of the image? The code below demonstrates how we can do this:

**Listing 4.3** `grab_and_set_region.py`

```
17 image = image[0:100, 0:100]
18 cv2.imshow("Image", image)
19
20 image[0:100, 0:100] = (0, 255, 0)
21
22 cv2.imshow("Updated", image)
23 cv2.waitKey(0)
```

On **Line 17** we grab a 100 x 100 pixel region of the image. In fact, this is the top-left corner of the image! In order to grab chunks of an image, NumPy expects we provide four

Main Menu

Next

Undo

Redo

okay

cancel

#### 4.3 ACCESSING AND MANIPULATING PIXELS

On **Line 14** we manipulate the top-left pixel in the image, which is located at coordinate (0,0) and set it to have a value of (0, 0, 255). If we were reading this pixel value in RGB format, we would have a value of 0 for red, 0 for green, and 255 for blue, thus making it a pure blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, not RGB format.

We actually read this pixel as 255 for red, 0 for green, and 0 for blue, making it a red color, not a blue color.

After setting the top-left pixel to have a red color on **Line 14**, we then grab the pixel value and print it back to console on **Lines 15 and 16**, just to demonstrate that we have indeed successfully changed the color of the pixel.

Accessing and setting a single pixel value is simple enough, but what if we wanted to use NumPy's array slicing capabilities to access larger rectangular portions of the image? The code below demonstrates how we can do this:

```

Listing 4.3: grabbing and setting
17 image = image[0:100, 0:100]
18 # cv2.imshow("Corner", corner)
19
20 image[0:100, 0:100] = [0, 255, 0]
21
22 # cv2.imshow("Updated", image)
23 cv2.waitKey(0)

```

On **Line 17** we grab a 100 x 100 pixel region of the image. In fact, this is the top-left corner of the image! In order to grab chunks of an image, NumPy expects we provide four

Main Menu

Next

Undo

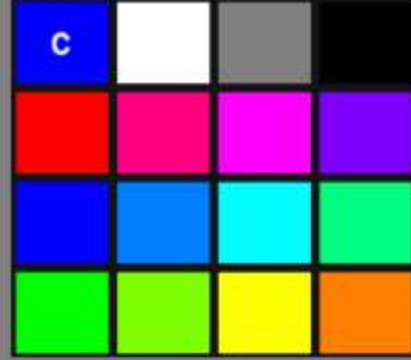
Redo

Highlight

Doodle

Transparency : 128

Brush Size : 90



On Line 14 we manipulate the to-be-left pixel in the image, which is located at coordinate (0, 0) and set it to have a value of (0, 0, 255). If we were reading this pixel value in RGB format, we would have a value of 0 for red, 0 for green, and 255 for blue, thus making it a pure blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, not RGB format.

We actually read this pixel as 255 for red, 0 for green, and 0 for blue, making it a red color, not a blue color.

After setting the top left pixel to have a red color on Line 14, we then grab the pixel value and print it back to console on Lines 15 and 16, just to demonstrate that we have indeed successfully changed the color of the pixel.

Accessing and writing a double-precision value is simpler enough.

but what if we wanted to use NumPy's array slicing capabilities to access larger rectangular portions of the image?

The code below demonstrates how we can do this:

Latino and Hispanic youth

```

77 output = image(0:100, 0:100)
78 cv2.imshow("output", output)
79
80 cv2.waitKey(100) = (k, 200, 0)
81
82 cv2.imshow("output", image)
83 cv2.waitKey(0)

```

On line 17 we grab a  $100 \times 100$  pixel region of the image. In fact, this is the top-left corner of the image! In order to grab chunks of an image, NumPy expects we provide four

**Main Menu**

**Next**

Undo

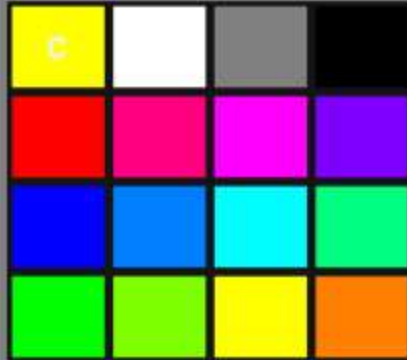
Redo

### Highlight

## Doodle

Transparency : 128

Brush Size : 90



#### 4.3 ACCESSING AND MANIPULATING PIXELS

On Line 14 we manipulate the top-left pixel in the image, which is located at coordinate (0,0) and set it to have a value of (0, 0, 255). If we were reading this pixel value in RGB format, we would have a value 0 for red, 0 for green, and 255 for blue, thus making it a pure blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, not RGB format.

We actually read this pixel as 255 for red, 0 for green, and 0 for blue, making it a red color, not a blue color.

After setting the top-left pixel to have a red color on Line 14, we then grab the pixel value and print it back to console on Lines 15 and 16, just to demonstrate that we have indeed successfully changed the color of the pixel.

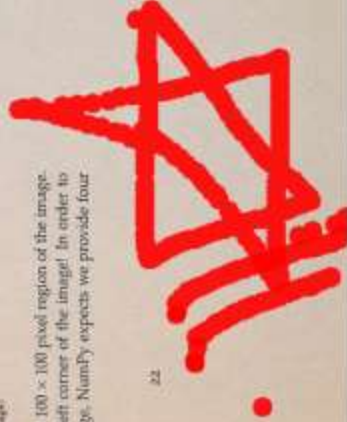
**Accessing and setting a single pixel value is simple enough,** but what if we wanted to use NumPy's array slicing capabilities to access larger rectangular portions of the image?

The code below demonstrates how we can do this:

```
Listing 4.3: grabbing and setting a pixel region  
17 image = image[0:100, 0:100]  
18 r = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  
19  
20 image[0:100, 0:100] = [0, 255, 0]  
21  
22 r = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  
23 r = r[0:100, 0:100]
```

On Line 17 we grab a 100 x 100 pixel region of the image. In fact, this is the top-left corner of the image! In order to grab chunks of an image, NumPy expects we provide four

22



Main Menu

Next

Undo

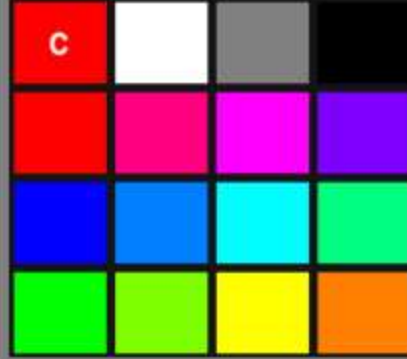
Redo

Highlight

Doodle

Transparency : 24

Brush Size : 15





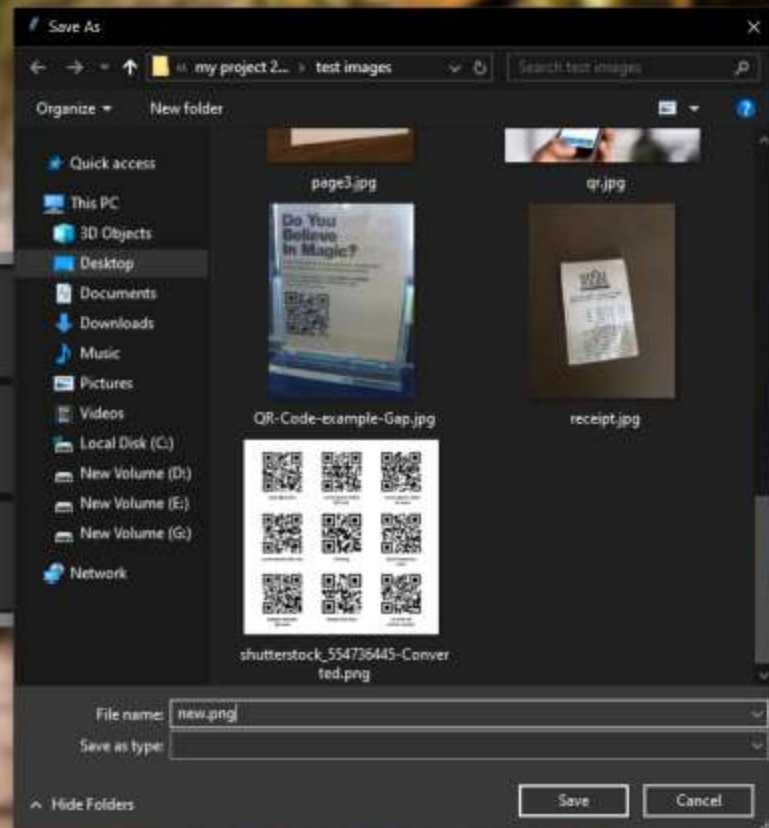
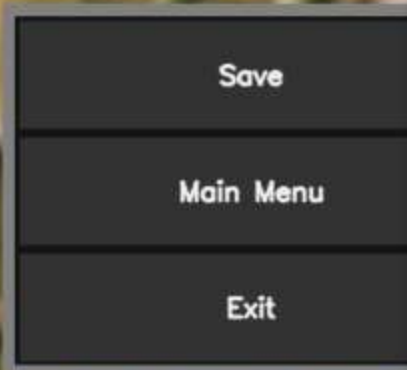
The background of the image is a blurred photograph of a dirt path in a forest. On either side of the path is a rustic wooden fence made of vertical posts and horizontal rails. The trees in the background are mostly bare, suggesting an autumn or winter setting. The overall color palette is muted, with browns, greys, and hints of green.

Save

Main Menu

Exit







Document Perspective Corrector

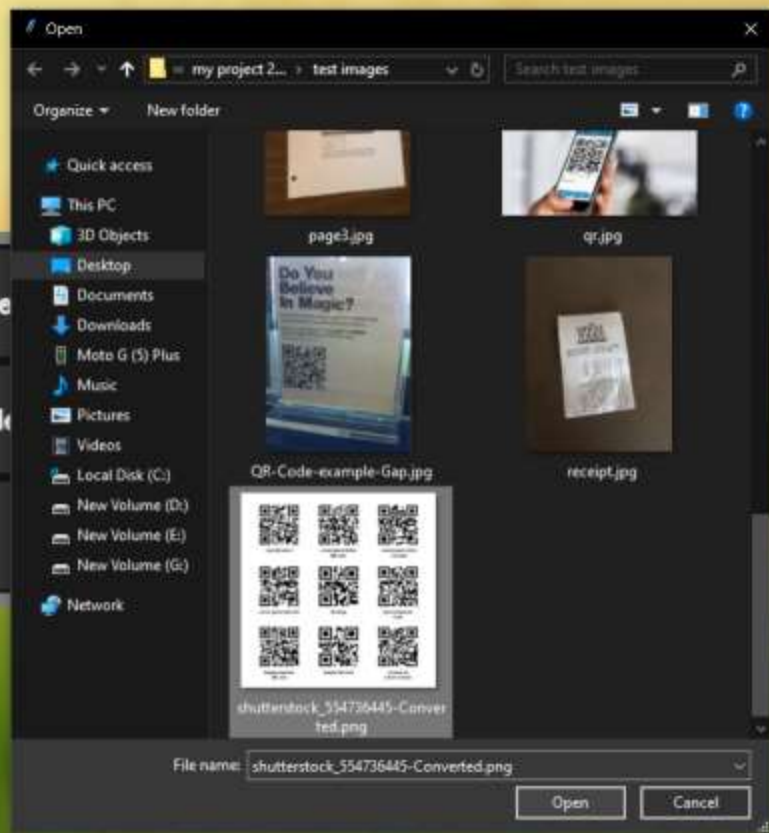
Image QR code detector

Exit

Document Perspective

Image QR code de

Exit





! just QR-code !



Lorem Ipsum Dolor  
QR-code



Lorem ipsum dolor  
sit amet



Lorem Ipsum QR-code



Nothing



Quick Response  
Code



Sample abstract  
QR-code



Sample QR-code



Ut enim ad  
minim veniam



! just QR-code !



Lorem Ipsum Dolor  
QR-code



Lorem ipsum dolor  
sit amet



Lorem Ipsum QR-code



Nothing



Quick Response  
Code



Sample abstract  
QR-code



Sample QR-code



Ut enim ad  
minim veniam



#### 4.3 ACCESSING AND MANIPULATING PIXELS

On Line 14 we manipulate the top-left pixel in the image, which is located at coordinate (0,0) and set it to have a value of (0, 0, 255). If we were reading this pixel value in RGB format, we would have a value of 0 for red, 0 for green, and 255 for blue, thus making it a pure blue color.

However, as I mentioned above, we need to take special care when working with OpenCV. Our pixels are actually stored in BGR format, **not** RGB format.

We actually read this pixel as 255 for red, 0 for green, and 0 for blue, making it a red color, *not* a blue color.

After setting the top-left pixel to have a red color on Line 14, we then grab the pixel value and print it back to console on Lines 15 and 16, just to demonstrate that we have indeed successfully changed the color of the pixel.

Accessing and setting a single pixel value is simple enough, but what if we wanted to use NumPy's array slicing capabilities to access larger rectangular portions of the image? The code below demonstrates how we can do this:

Listing 4.3: Getting and setting

```
17 corner = image[0:100, 0:100]
18 cv2.imshow('Corner', corner)
19
20 image[0:100, 0:100] = (0, 255, 0)
21
22 cv2.imshow('Updated', image)
23 cv2.waitKey(0)
```

On line 17 we grab a  $100 \times 100$  pixel region of the image. In fact, this is the top-left corner of the image! In order to grab chunks of an image, NumPy expects we provide four

